Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Алтайский государственный технический университет

им. И.И. Ползунова»

Факультет (институт) Информационных технологий

Кафедра Прикладная математик

Отчет защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.М. Старолетов

(подпись преподавателя) (инициалы, фамилия)

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2017 г.

Отчет

по лабораторной (практической) работе № 3

по дисциплине Архитектурное проектирование и паттерны программирования

(наименование дисциплины)

ЛР 09.03.04.10.000 ОТ

(обозначение документа)

Студенты группы ПИ-42 Е.А. Колесников

(инициалы, фамилии)

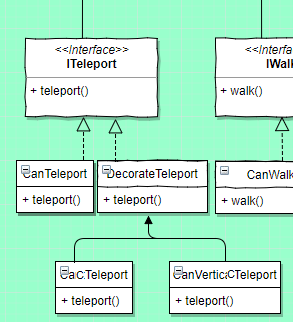
Преподаватель Доцент к. канд. физ.-мат. наук С.М.Старолетов

(должность, ученое звание) (инициалы, фамилия)

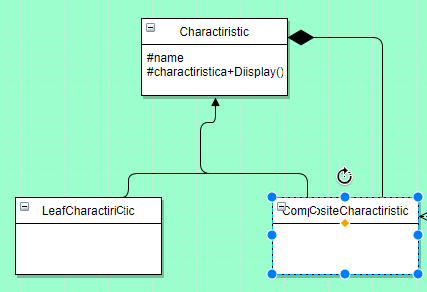
Барнаул 2017

**Задание:** реализовать паттерны: Адаптер, Декоратор, Итератор, Композит

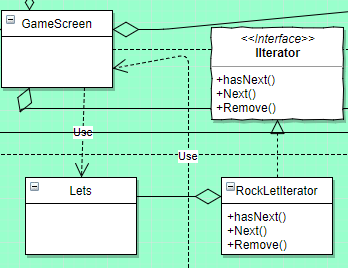
Декаратор:



Композит:



Итератор:



**Код:**

public interface IIterator {  
 public boolean hasNext();  
 public RockLet Next();  
 public void Remove(RockLet rockLet);  
}

public class Lets {  
 private ArrayList<RockLet> rockdrop;  
 private long lastDropTime;  
  
 public Lets() {  
 lastDropTime=0;  
 rockdrop=new ArrayList<RockLet>();  
 }  
  
 public ArrayList<RockLet> getRockdrop() {  
 return rockdrop;  
  
 }  
  
 public void setRockdrop(ArrayList<RockLet> rockdrop) {  
 this.rockdrop = rockdrop;  
 }  
  
 public void Add\_Rock()  
 {  
  
 if(TimeUtils.*nanoTime*()-lastDropTime>1000000000) {  
 // RockLet rock = new RockLet();  
 // rockdrop.add(rock);  
 lastDropTime=TimeUtils.*nanoTime*();  
 }  
  
 }  
  
}

public class RockIterator implements IIterator {  
 private Lets lets;  
 private int currentlet;  
 boolean letIsIterated;  
  
 public RockIterator(Lets lets) {  
 this.lets = lets;  
 currentlet=0;  
 letIsIterated=false;  
 }  
  
 @Override  
 public boolean hasNext() {  
 if(lets.getRockdrop().size()>currentlet)  
 return true;  
 else  
 return false;  
  
 }  
  
 @Override  
 public RockLet Next() {  
 RockLet rockLet ;  
 rockLet=(RockLet)lets.getRockdrop().get(currentlet);  
 currentlet++;  
 return rockLet;  
 }  
  
 @Override  
 public void Remove(RockLet rockLet) {  
 if(lets.getRockdrop().size()>0) {  
 lets.getRockdrop().remove(rockLet);  
 }  
 }  
}

**Композит**

public abstract class Characteristics {  
 protected String name;  
 protected String characteristica;  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getCharacteristica() {  
 return characteristica;  
 }  
  
 public void setCharacteristica(String characteristica) {  
 this.characteristica = characteristica;  
 }  
  
 public Characteristics() {  
  
 }  
  
 public Characteristics(String name, String characteristica) {  
 this.name = name;  
 this.characteristica = characteristica;  
 }  
 public abstract String Display();  
}

public class CompositeCharacteristic extends Characteristics {  
 private ArrayList<Characteristics> characteristics;  
 public CompositeCharacteristic() {  
 super();  
 characteristics= new ArrayList<Characteristics>();  
 }  
  
 public CompositeCharacteristic(String name, String characteristica) {  
 super(name, characteristica);  
 }  
 public void add(Characteristics characteristics)  
 {  
 this.characteristics.add(characteristics);  
 }  
 @Override  
 public String Display() {  
 String value="";  
 Iterator<Characteristics> iterator =characteristics.iterator();  
 while (iterator.hasNext()) {  
 Characteristics ch = iterator.next();  
 value+=ch.name + ": " + ch.characteristica+"\n";  
 }  
 return value;  
 }  
}

public class LeafCharacteristic extends Characteristics {  
 public LeafCharacteristic() {  
 super();  
 }  
  
 public LeafCharacteristic(String name, String characteristica) {  
 super(name, characteristica);  
 }  
  
 @Override  
 public String Display() {  
 return name + ": " + characteristica;  
 }  
}

**Декоратор:**

public abstract class DecoratorTeleport implements ITeleport {  
 protected ITeleport teleport;  
  
 public DecoratorTeleport(ITeleport teleport) {  
 this.teleport = teleport;  
 }  
  
 @Override  
 public void teleport(Entity e, int orientation) {  
  
 }  
}

public class CanTeleport implements ITeleport {  
 private int velocityX;  
 private int velocityY;  
  
 @Override  
 public int getVelocityX() {  
 return velocityX;  
 }  
  
 @Override  
 public void setVelocityX(int velocityX) {  
 this.velocityY=velocityX;  
 }  
  
 @Override  
 public void setVelocityY(int velocityY) {  
 this.velocityY=velocityY;  
 }  
  
 @Override  
 public int getVelocityY() {  
 return velocityY;  
 }  
  
 public CanTeleport() {  
 velocityX=400;  
 velocityY=0;  
 }  
 public CanTeleport(int velocity) {  
 this.velocityX = velocity;  
 }  
 @Override  
 public void teleport(Entity e, int orientation) {  
 e.setPosition(new Vector3(e.getPosition().x + velocityX \* orientation, e.getPosition().y+velocityY, 0));  
 }  
}

public class CantTeleport extends DecoratorTeleport {  
  
 public CantTeleport(ITeleport teleport) {  
 super(teleport);  
 }  
  
 @Override  
 public void teleport(Entity e, int orientation) {  
  
 }  
  
 @Override  
 public int getVelocityX() {  
 return 0;  
 }  
  
 @Override  
 public int getVelocityY() {  
 return 0;  
 }  
  
 @Override  
 public void setVelocityX(int velocityX) {  
  
 }  
  
 @Override  
 public void setVelocityY(int velocityY) {  
  
 }  
  
}

public class CanVerticalTeleport extends DecoratorTeleport {;  
  
 public CanVerticalTeleport(ITeleport teleport, int velocity) {  
 super(teleport);  
 }  
  
 public CanVerticalTeleport(ITeleport teleport) {  
 super(teleport);  
 }  
  
 @Override  
 public void teleport(Entity e, int orientation) {  
 int vel=teleport.getVelocityX();  
 teleport.setVelocityX(0);  
 teleport.setVelocityY(vel);  
 teleport.teleport(e,orientation);  
 e.onGround=false;  
 }  
  
 @Override  
 public int getVelocityX() {  
 return 0;  
 }  
  
 @Override  
 public int getVelocityY() {  
 return 0;  
 }  
  
 @Override  
 public void setVelocityX(int velocityX) {  
  
 }  
  
 @Override  
 public void setVelocityY(int velocityY) {  
  
 }  
}

**Тесты:**

Композит:



Декоратор:



Итератор:

